1999-03-01

Implementing a decision support system on the World Wide Web

Protacio, Patrick E.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/8510

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

IMPLEMENTING A DECISION SUPPORT
SYSTEM ON THE WORLD WIDE WEB

by

Patrick E. Protacio

March, 1996

Thesis Advisor: Hemant K. Bhargava

Approved for public release; distribution is unlimited.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE IMPLEMENTING A DECISION SUPPORT SYSTEM ON THE WORLD WIDE WEB			5. FUNDING NUMBERS
6. AUTHOR(S) Protacio, Patrick E.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

As the popularity and use of the World Wide Web (the Web) increases daily, many technologies and applications that were initially developed and used as stand alone tools are migrating towards the Web. Decision support technologies are examples of applications that traditionally have been developed as stand alone systems. One feature common to all decision support systems is that they require user input to produce results. However, due to some inherent limitations with the Web, some modifications must be made to allow users to interact with these technologies. Common Gateway Interface (CGI) programs, or scripts, provide Web servers the capability to produce dynamic documents and maintain user information or state. With CGI scripts, developers can capture user entered data, access external applications to process the data, and return the results. Using CGI scripts to interface with database applications gives Web servers the ability to maintain state by tracking user data, information, and preferences. This thesis implements a Windows spreadsheet based decision support system (DSS) designed to optimize radar coverage of tactical aircraft. Using Delphi to write CGI scripts, the Web based DSS allows the user to enter data to run a new problem, save input and output data, and retrieve the saved data at a later time.

14. SUBJECT TERMS Decision Support System, DSS, World Wide Web, CGI			15. NUMBER OF PAGES 60
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

IMPLEMENTING A DECISION SUPPORT SYSTEM
ON THE WORLD WIDE WEB

Patrick E. Protacio

Lieutenant Commander, Supply Corps, United States Navy

B.S., United States Naval Academy, 1983

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY
MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
March 1996

ABSTRACT

As the popularity and use of the World Wide Web (the Web) increases daily, many technologies and applications that were initially developed and used as stand alone tools are migrating towards the Web. Decision support technologies are examples of applications that traditionally have been developed as stand alone systems. One feature common to all decision support systems is that they require user input to produce results. However, due to some inherent limitations with the Web, some modifications must be made to allow users to interact with these technologies. Common Gateway Interface (CGI) programs, or scripts, provide Web servers the capability to produce dynamic documents and maintain user information or state. With CGI scripts, developers can capture user entered data, access external applications to process the data, and return the results. Using CGI scripts to interface with database applications gives Web servers the ability to maintain state by tracking user data, information, and preferences. This thesis implements a Windows spreadsheet based decision support system (DSS) designed to optimize radar coverage of tactical aircraft. Using Delphi to write CGI scripts, the Web based DSS allows the user to enter data to run a new problem, save input and output data, and retrieve the saved data at a later time.

TABLE OF CONTENTS

I. INTRODUCTION

As the popularity and use of the World Wide Web (the Web) increases daily, many technologies and applications that were initially developed and used as stand alone tools are migrating towards the Web. This shift to a client-server paradigm not only reduces distribution and procurement costs, it also provides an opportunity for more clients to access and utilize these formerly isolated technologies and applications. This thesis describes the implementation of a stand alone decision support system (DSS) on the Web.

A.   DSS BACKGROUND

A prototype DSS, the Simplified Aircraft Radar Coverage Scheduling System (SARCSS), was designed and developed to provide U. S. Navy decision makers a means of effectively assigning aircraft to cover a given amount of airspace while minimizing costs. SARCSS attempts to:

- Optimize the allocation of carrier-based, fixed-wing aircraft to ensure 100% coverage of a given area.

- Minimize fuel and operating costs.

- Evaluate multiple alternatives.

The program uses a multi-attribute approach combined with a graphical user interface. It also has ad-hoc query capabilities that can accommodate limited "what if" scenarios when outside events further constrain the problem. SARCSS will ultimately reduce the

man-hours currently required to manually solve these radar coverage problems and improve the accuracy of the results. As a prototype, SARCSS has some specific limitations and is applicable only in a narrow range of possible scenarios. Given that certain assumptions were made, the model is reliable and provides decision makers with a valuable planning tool. (Braley, 1995)

SARCSS is a spreadsheet based DSS and was developed using Quattro Pro v5.0 for Windows™. A stand alone program, it was designed to be easily installed on any IBM compatible microcomputer currently in use throughout the Navy. Using mathematical formulas and Quattro Pro optimizers, SARCSS prompts users for variable data input through the use of dialog boxes and results are displayed on a full spreadsheet page. The program also has the capability of producing graphs depicting any user performed sensitivity analysis. The ad-hoc capability allows users to change any constants that will have an effect on the final results. Finally, users have the capability for saving their problems for later retrieval, further reducing the time spent entering parameters, and allowing them to compare old results to new ones.

B.   WHY THE WEB?

Although SARCSS was designed to be portable, it still has its limitations. For example, it can only run with Quattro Pro for Windows and on an IBM compatible computer. Implementing SARCSS on the Web provides numerous advantages. It would provide users world-wide, 24 hour, real time and interactive access to the program. Users would no longer be limited to an IBM compatible platform since there is no longer any requirement to have the software locally. The only requirement would be to have a

2

platform specific Web browser capable of interpreting SARCSS files. Not only does this free local computer resources, it also reduces costs. There is no longer any need to purchase the spreadsheet application required to run the program and no need for local program maintenance. Development costs would also be reduced. Version improvements would only have to be made in one place, eliminating the requirement to send numerous floppy disks to all users.

C.   DECISION TECHNOLOGIES ON THE WEB

There are a number of technologies currently on the Web that provide decision support. A very simple example is the Tire Selector and Store Locator offered by the Goodyear Tire and Rubber Company. The Tire Selector provides users with the types of tires available for a particular vehicle, and the Store Locator lists stores based on a user entered zip code or city and state. (Goodyear, 1995) Additionally, there are a number of operations research technologies based on mathematical programming that are available on the Web. Netlib is a repository of mathematical software, such as approximation algorithms and optimization software. It was originally developed to distribute this software via electronic mail but is now available on the Web. The National Institute for Standards and Technology maintains a library of decision technologies called GAMS (Guide to Available Mathematical Software). An electronic yellow pages, GAMS catalogs several hundred models and algorithms. For each of these models and algorithms, GAMS provides the user with a list of available products along with certain metainformation, such as location of the software, how to access it, and what type of platform it runs on. Finally, the Optimization Technology Center (OTC) maintains a

3

comprehensive collection of computational resources pertaining to mathematical programming. One of these resources is the NEOS (Network Enabled Optimization System) guide, which provides users with an organized and searchable collection of information about mathematical programming. (Bhargava, Krishnan, Muller, 1996)

1. Using DecisionNet

Research and development is currently being conducted involving a central repository of decision support systems or technologies available over the Web. Known as DecisionNet, decision support application developers or providers would have the opportunity to register their applications with DecisionNet, which would then provide users with a database of decision support technologies available for use over the Web. Instead of users linking to the actual location of the technology, DecisionNet, as part of this centralized service, would act as an mediator between the technology, which would reside on the provider's server, and the user. (Bhargava, King, McQuay, 1995)

However, before a viable interface between a technology and DecisionNet can be developed, there must be information on how an application is implemented on the Web. DecisionNet must know various information such as what data types are used, how data is entered into the technology, and how results are returned to the user. This thesis will provide some of the information necessary for DecisionNet to properly capture and transmit user data, and retrieve and display results from the technology.

D. THESIS ORGANIZATION

Chapter II will discuss the Web environment and current Web technologies. Chapter III will cover decision technologies and tools available on the Web that are useful

for developing or implementing a technology on the Web. SARCSS implementation on the Web will be discussed in Chapter IV. Conclusions and recommendations for further research will be covered in Chapter V.

II. WORLD WIDE WEB TECHNOLOGIES

A.   CLIENT-SERVER ENVIRONMENT

In a conventional client-server environment, processing capabilities are distributed throughout the network. The client usually initiates a request for processing or some other type of service from the server. The server processes or performs the service requested by the client. In many cases, the client and server share processing duties to optimize application efficiency and effectiveness. The client performs processes associated with the applications that it has locally, while the server performs processing tasks in support of the clients. In this environment, clients may frequently download data from servers for local processing and have the flexibility of interacting with more than one server. (Long, 1994)

In the context of the Web, in order to view or process any information or data provided by a server, all clients must have a Web browser, a platform specific application, which can process server information. The normal Web client-server conversation follows this sequence:

- The client requests data or information from the server.

- The server decides exactly what the client is requesting and then sends the data.

- The client then takes action on the data received.

Once the client receives the data, it must decide how it will handle it. For "standard" (which will be defined in the next section) information, the client's browser will take care of the information. For specialized information or data, one of three actions can take place:

- A "helper" application will be called.
- The browser will request the user to associate an application to the data.
- The data can be saved to a disk for later access/use.

Despite any action taken by the client, the server does most of the processing. It must retrieve the data and send it to the client. The client, on the other hand, performs limited processing by taking the data and passing it to another application for further processing, if required. However, there is no real time, interactive processing between the client and the server. If a client requests and receives an executable file, it must have the appropriate hardware and software for working with that executable file.

B.   STANDARDIZED MARKUP LANGUAGE

What links all clients and servers on the Web is a standard language for writing Web documents. Known as the Hypertext Markup Language, or HTML, it provides virtually anyone the capability of creating a Web page of information. Through hypertext, any Web document can be linked to any other document on any server in the network. A markup language provides document creators the ability to focus on the format of the document, which emphasizes such things as headings, paragraphs, and images, and not so much on its appearance, where things like typeface, font size and style

are of more concern. By focusing on a document's structure, a markup language proves to be a simple and portable way of allowing any client using any operating system, even a dumb terminal, to view a document through a browser with relatively little loss of information. Although a standard language such as HTML contributes to universal authorship, it has its drawbacks.

C.   WEB WEAKNESSES

Static pages are one of the drawbacks of an HTML document. When a Web document is created, it is usually saved to an area on a disk that a server can access. When a client sends a request to the server to retrieve the document, the server will always send the same document. No matter who sends the request or at what time the request was sent, the same, static document is returned by the server and displayed on the client's browser. Although there are many circumstances where this may be useful or desirable, it can be restrictive and does not allow any interaction between the client and the server. With only browsable content, client-server sessions on the Web are limited to the preconceived scenarios of the Web author (Gonzales, 1996).

The other main drawback of the Web is its lack of ability to maintain state information about a connection. A Uniform Resource Locator (URL) is what a client uses to navigate around the Web. In general, a URL is composed of three parts: the protocol/program/action to perform, location of the machine, and path of the resource, such as the Web server name along with the file location on that server (Lynnworth, 1996). Whenever a client makes a connection to a Web server, which occurs every time a URL on the server is accessed, the server has no way of correlating the new URL

9

request to any previous ones. To a Web server, one client's URL request is indistinguishable from another client's request. Furthermore, multiple requests from any one client are completely unrelated to each other. Once the server has provided the information or services the client has requested, the connection is terminated, leaving no trace of information about the request. Server activity logs keep track of user information but these logs are only for system administration and cannot be seen by the Web server software. Because the server does not keep track of past connections and requests, the server has no memory about what the client has requested or done, and cannot build state information for a client. (Gonzales, 1996)

III. PROVIDING DECISION TECHNOLOGIES ON THE WEB

A.   CHARACTERISTICS OF DSS TECHNOLOGIES

Decision modeling, decision theory, and decision analysis attempt to create models which can assist decision makers in selecting the most appropriate course of action to solve a particular problem. These models usually involve some mathematical computation to produce an output. Sprague and Carlson define decision support systems as "computer based systems that help decision makers confront ill-structured problems through direct interaction, with data and analysis models." They go on to create a dialog, data, and modeling paradigm which consists of dialog between the user and the system, data that supports the system and the models that provide the analysis. Although the components may be different in each application, they always exist in some form. Finally, DSS's can range from "simple ad hoc analysis" that could be considered an end user task to large, complex systems that may have many attributes of a major system application. (Sprague, McNurlin, 1993)

Data used by a DSS may be internal to the system or the DSS may be required to access an external database. Similarly, computations may require external applications such as a spreadsheet. A DSS should also have the capability to save results either for further analysis or comparison with previous or future problems. Finally, a DSS should have the capability of adjusting to users or problems according to previously selected or predetermined options or parameters.

B.   ENABLING TECHNOLOGIES

Because of the dynamic nature of a DSS, implementing one on the Web requires a special tool to overcome the two problems discussed earlier about the Web in general - static pages and no user interaction. A Web server is often used as a gateway to some sort of information system that usually consists of documents and possibly a database. The Common Gateway Interface (CGI) is an agreement between Web server implementors about how to integrate gateway scripts and programs (World Wide Web Consortium, 1996).   The National Center for Supercomputing Applications (NCSA) Software Development Group currently maintains the CGI specifications. It defines CGI as "a standard for interfacing external applications with information servers, such as ... Web servers." (NCSA, 1996) This interface between the Web server and other applications is accomplished through any executable program that conforms to CGI rules about how input and output is treated (Lynnworth, 1996). CGI programs (usually called scripts) can be written in any language that can create an executable file for a given operating system. Unlike a static HTML file, a CGI program is executed in real time and can output dynamic information.

Although the NCSA specification is mainly written for the UNIX platform capable of running C or Perl executable files, Robert Denny has amended the CGI specification for use in the Windows environment. (Denny, 1995) In his specification, he states that instead of using "stdin" and "stdout" to stream data in and out of the CGI script, as is done in a UNIX operating environment, one would use a set of files in a temporary directory to hold the information. There would be a set of two or more files defining the

12

client's request, and the CGI program would be responsible for creating a third "output" file before terminating. Except for the requirements for accepting input and returning output, "scripts can do almost anything". (Lynnworth, 1996)

Using CGI, script developers must define how input from the user will be made available to the executable program. All output that will be returned to the user must be in HTML format so that the client's browser can display it. Output can either direct the user to a previously created static document or the script can generate the output on the fly.

CGI scripts or programs are very powerful Web tools since they allow two way interaction between the Web client and server. They can obtain information from users and act on that information, producing variable results. Depending on the platform, there are any number of scripting languages available. Of all the languages available in the Unix environment, Perl and C seem to be used most often. Perl has also been developed for DOS and Windows (32 bit) operating systems. However, its use is fairly limited, due partly to its newness, and more likely to the established base of other languages in the personal computer (PC) environment. In the microcomputer world, Basic, Visual Basic, and C appear to be the most popular scripting languages, with Delphi gaining considerable attention and use.

Clearly, scripts help to overcome the problem of static documents and allow user interaction. But they also provide a way of saving state. Recall that saving state meant keeping track of client connections and requests, as well as client preferences. Most of the current development with CGI scripts focuses on capturing user entered data,

information, or requests and storing that information in either a file or database. By constructing a script to access these files or databases each time a user accesses or makes a request to a Web server, the script and server can tailor its output for each individual user. With these types of CGI scripts, a Web server has the capability of distinguishing users and keeping track of all connections and requests.

IV. IMPLEMENTING SARCSS ON THE WEB

A. DESIGNING A WEB INTERFACE

The goal in designing a Web interface for SARCSS was to emulate the various screens and user input dialog boxes used in the stand alone program. The functions developed for Web implementation focused on allowing a user to enter data to solve a new problem and allowing the user to retrieve any previous work. Using HTML, an introduction page was created, followed by a menu page which provides the user with a number of options (Figures 1 and 2).
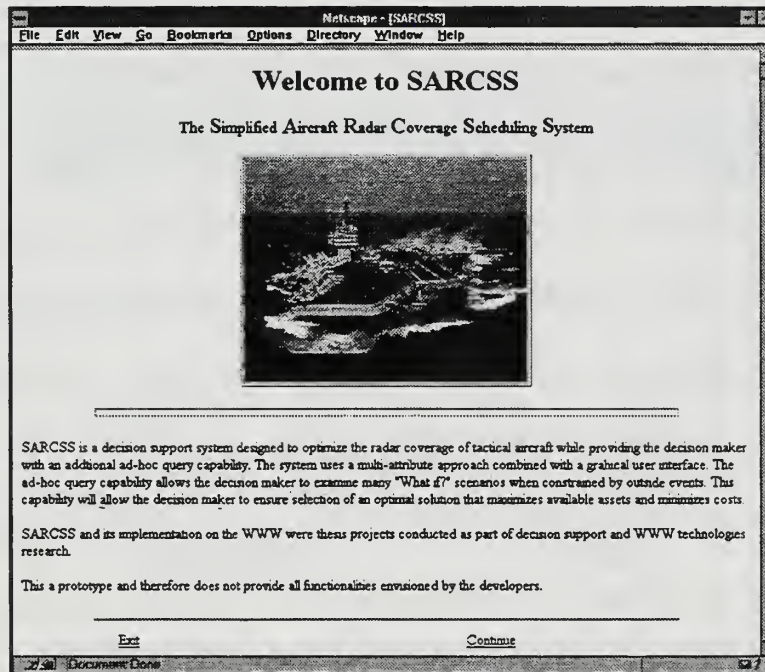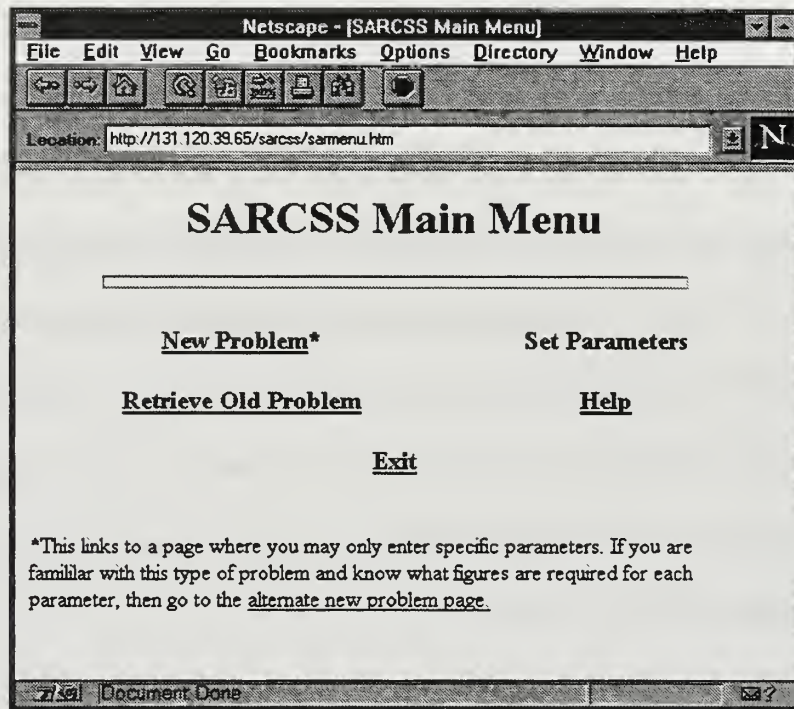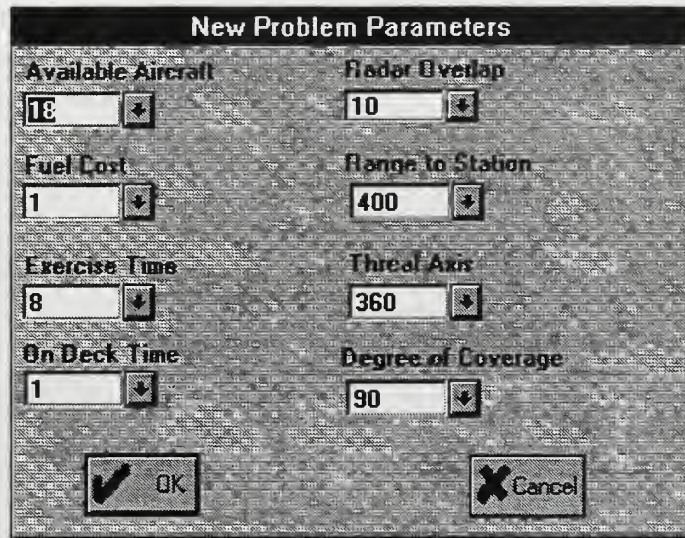

Figure 1. SARCSS Web Welcome Page

Figure 2. SARCSS Main Menu Web Page

Next, the user input pages had to be designed. The stand alone version's dialog boxes were used as templates for designing the Web pages. Figure 3 shows the dialog box used in the stand alone version to enter data for problem solving. This interface easily translated into an HTML form, which can be seen in Figure 4. Because the parameters to SARCSS are unlikely to change with any sort of frequency, the HTML page is static, instead of being created by a CGI script.

Although the drop down option selection feature was retained, this greatly limits the user only to the options provided on the list. To provide more user flexibility in entering parameters, a second new problem page was created without the drop down options. The main menu page, Figure 2, has a note below the menu options where the user may select this alternative page. Since many users would have access to SARCSS
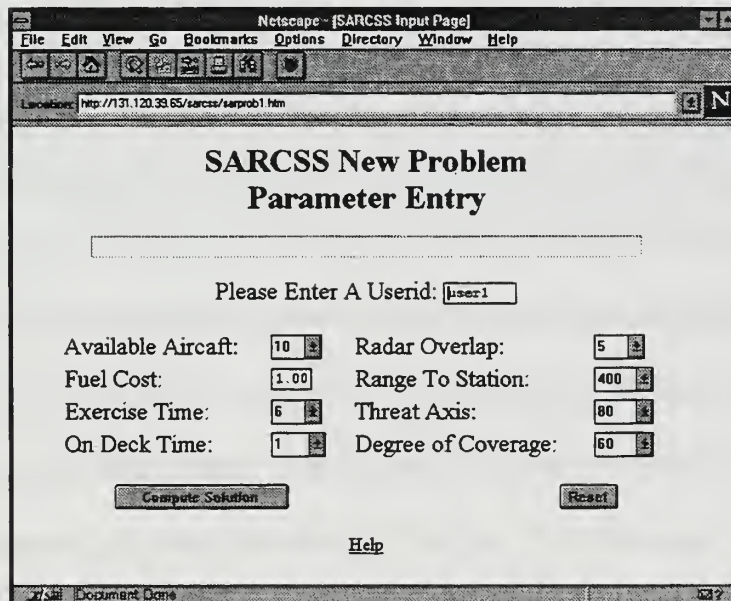
16

on the Web, a capability for keeping track of their saved work was needed. This is the reason for the Userid field on the form.

Figure 3. Typical SARCSS Input Dialog Box

Figure 4. SARCSS New Problem Parameter Entry Web Page

B. ACCESSING EXTERNAL APPLICATIONS

Once the user entered all the required data, a method was needed to capture the data and pass it to the spreadsheet for processing. First I created a temporary Paradox database table. The table has fields for all the user entered data and two additional items, the computer's date and time. Using DDE, each field in the table is linked to the appropriate cell in the spreadsheet where the data is required. The userid, date and time were linked to empty cells. The table is temporary in the sense that it only contains one record, which is overwritten each time a user submits data for processing.

A Delphi CGI script was written to handle the process of passing data from the HTML form to the temporary table (Pascal code for the scripts are in the Appendicies). The script makes use of CGI component features and other features normally found in Delphi, such as DDE. The CGI component requires each parameter entered through the form to be assigned as string variables. The script then opens the table, deletes the single record, appends the new information, and finally closes the table. The CGI function of the script is initially limited to retrieving the data and putting it into the table. It will come into play again towards the end of the script. However, the rest of the script uses features that are inherent in Delphi.

Once the data is in the spreadsheet, some manipulation is done with the results prior to passing them back to the script. First, on a separate page within the spreadsheet, I wrote the HTML output file. Figure 5 shows how various cells on this page contained the actual HTML tags and the text that would be displayed on the output page. It also shows that the results, which were computed on a separate page were linked to

18

appropriate cells on the output page. Again, once the results were computed, they were dynamically updated on the output page.

```
Quattro Pro for Windows - WORKDSS1.WB1
 File   Edit   Block   Data   Tools   Graph   Property   Window   Help

1   <html><head><title>SARCSS Problem Results</title></head>
2   <body BGCOLOR="#FFFFFF"></body><Form Method=Post Action="http://131.120.39.65/cgi-win/sarcss/sarcsave.exe">
3   <center><h1>Problem Results</h1></center><hr><br>
4   <Center><table border=0 width=400><tr><th>Userid</th><th>Date</th><th>Time</th></tr>
5   <tr><td align="center"><INPUT Nan user1       " rows=1 size="12"></td>
6   <td align="center"><INPUT Name="  2/27/96     " rows=1 size="10"></td>
7   <td align="center"><INPUT Name="  4:20:43 PI " rows=1 size="13"></td></tr></table></Center><br><br><p>
8   <Center><table border=0 width=600><Caption><Font Size=+3><b>Input Parameters</b></Font></Caption><tr><td><Font Siz
9   <td><INPUT Name="AvailAC" Type=      16  " rows=1 size="6"></td><td><Font Size=+1>Radar Overlap: </font></td>
10  <td><INPUT Name="RadarOvrlap" T      10  " rows=1 size="6"></td></tr>
11  <tr><td><Font Size=+1>Fuel Cost: </font></td>
12  <td><INPUT Name="FuelCost" Type      1   " rows=1 size="6"></td><td><Font Size=+1>Range To Station: </font></td>
13  <td><INPUT Name="RngToSta" Typ      400 " rows=1 size="6"></td></tr>
14  <tr><td><Font Size=+1>Exercise Time: </font></td>
15  <td><INPUT Name="ExTime" Type=      10  " rows=1 size="6"></td><td><Font Size=+1>Threat Axis: </font></td>
16  <td><INPUT Name="ThrtAxis" Type      100 "rows=1 size="6"></td></tr>
17  <tr><td><Font Size=+1>On Deck Time: </font></td>
18  <td><INPUT Name="ODTime" Type       1   " rows=1 size="6"></td><td><Font Size=+1>Degree Of Coverage: </font></td>
19  <td><INPUT Name="DegOfCov" Typ      80  " rows=1 size="6"></td></tr></Table><br><br><p>
20  <table border=0 width=600><Caption><Font Size=+3><b>Problem Results</b></Font></Caption><tr><td><Font Size=+1>Air
21  <td><INPUT Name="ACReqd" Type       12  " rows=1 size="6"></td><td><Font Size=+1>Time On Station: </Font></td>
22  <td><INPUT Name="TOnSta" Type=      1.4 " rows=1 size="6"></td></tr>
23  <tr><td><Font Size=+1>FEBA Range: </Font></td>
24  <td><INPUT Name="FEBARng" Typ       726 " rows=1 size="6"></td><td><Font Size=+1>Flight Time per Aircraft: </font></
25  <td><INPUT Name="FltTime" Type=      3.3 " rows=1 size="6"></td></tr>
26  <tr><td><Font Size=+1>Range Of Coverage: </Font></td>
27  <td><INPUT Name="RngOfCov" Typ      520 " rows=1 size="6"></td><td><Font Size=+1>Total Fuel Cost: </font></td>
28  <td><INPUT Name="TotFuelCost" T     60896 " rows=1 size="10"></td></tr></table></Center><br><br>
29  <Center><INPUT Type="SUBMIT" Value="Save This Problem"><br><br></center></Form>
30  <body><Center><Table border=0 width=400><tr><td align="left"><b><a href="http://131.120.39.65/sarcss/sarprob1.html">New
31  <td align="left"><a href="http://131.120.39.65/sarcss/sarmenu.htm">Main Menu</A></td><td align="right"><b><a href="bharga
32  <b>*</b> If you do not save your work before starting a new problem, all data will be lost.</body></html>
33
```

Figure 5. SARCSS Spreadsheet Output Page

When the ouput page update was complete, the script used the Delphi DDE feature to link to each cell that contained any sort of data - HTML, text, and results - on the output page. Each DDE item was then written to a text file with a .HTM extension. Since Delphi cannot automatically create files on its own, the text file was manually created prior to writing the script. The file can also be considered a temporary file since each time a new problem is executed, the data in the file gets overwritten. The output file is just an ASCII text file version of the spreadsheet page seen in Figure 5.

19

C.   DISPLAYING RESULTS ON THE WEB

Because the script does not return results directly to the client, the user must have a way of accessing the output file. This is where the CGI components come back into play. Prior to script completion, an interim HTML page is created on the fly that contains a hypertext link to the output page (Figure 6). Note that the results page (Figure 7) is actually an HTML form. This provides the capablility for saving results if the user desires. If the user elects to save the problem, two separate tables are used, one for input data, the other for ouput data. In each table, the userid, date, and time are key fields.

Figure 6. SARCSS New Problem Interim Web Page

Figure 7. SARCSS New Problem Results Web Page

D.   RETRIEVING PREVIOUS WORK

If a user has done any previous work, SARCSS provides an option for retrieving the information and displaying both input and output data, essentially browsing the database for all records pertaining to a certain user. To do this, another Delphi CGI script was used. In addition to the main CGI component, a CGI database component was needed to do the actual drawing of the table. To retrieve a problem, the script requires a userid. This userid is first entered into a temporary table. Again, this table contains only one record which is overwritten each time a userid is submitted. The script then takes the userid and performs an SQL query on both input and output tables. Since the userid is dynamic, the query is done on the fly. The query will retrieve all records in the

21

tables that correspond to the userid. The temporary table containing the query's userid is needed for aesthetic reasons. The userid only needs to be displayed once on the output page. However, if either one of the tables where the query is performed is used to display the userid, that field will be displayed the same number of times the user has problems saved. Finally, the script creates an output HTML page on the fly, drawing all three tables - userid, input, and output (Figure 8).

Figure 8. SARCSS New Problem Results Web Page

V. CONCLUSIONS

This thesis took a DSS that was initially designed as a stand alone system and implemented it on the World Wide Web. It shows that using a number of tools, a Web server has the capability of accessing and executing various external applications. It also shows that CGI programs do not have to be limited to interface only with database applications. However, there are still areas that need futher development and research.

A.   CURRENT LIMITATIONS AND INEFFICIENCIES

Despite the numerous functions available on the stand alone version, the DSS, as it is currently implemented on the Web, provides very limited functionality. In the Web version, the user can only run a new problem and view previous work. The stand alone version, however, provides additional functions such as sensitivity analysis and graphical charts.

Additionally, the methods used in the current implementation to produce the results on the Web are not as efficient as possible. The current method of producing the output file which displays a problem's results involves having to pass many small pieces of data into a file and creating an intermediate HTML page. Finally, the Web version is incapable of handling more than one user at a time since certain data is overwritten each time a user performs some work.

23

B.   FURTHER RESEARCH

There are a number of areas where further research and development could be accomplished. These areas can focus on development related to this specific DSS, research into other DSSs and the use of agents to act as intermediaries between the DSS and the Web user.

1.   Related Development

Web implementation of SARCSS's sensitivity analysis and graphical charts are two areas that need development.   Implementing sensitivity analysis should prove to be similar to what has already been accomplished since it mainly requires running a spreadsheet macro and displaying results on the Web. Implementing the graphics would also be similar since Quattro Pro allows users to export graphics into another file. Another useful feature is that the graph can be saved into a .GIF format, which most Web browsers can display. Again, a Quattro Pro macro would have to be created and then executed by a CGI script.

The next area for development could focus on a more efficient method of displaying a problem's results. Having to individually link each cell in the speadsheet that contains output data to the CGI program and then having to write each linked item to the output file is not the most practical method for producing results for display on the Web. By developing a method for directly displaying results on a Web browser, the interim page created by the CGI script, and which links to the results page, could be eliminated.

Multi-client, simultaneous use is another area for future development. This process would require a method for tracking current users and dynamically naming temporary files and tables with a user's identification. The final area for development could focus on distributed applications. Could the spreadsheet, database and CGI script be placed on different servers instead of residing on the same server, as they currently do? If there is a high demand for the DSS, this would help distribute the workload between three servers, instead of slowing down one server with all the applications.

2. Other Decision Support Systems

Although SARCSS was developed using a spreadsheet application, other DSSs make use of any number of languages and applications. The majority of these systems are currently stand alone versions. Implementing them on the Web may require different processes than the ones used for SARCSS. With current technology, CGI programs and database connectivity would be the common factors. Depending on the DSS application, connectivity to a database could prove to be quite challenging.

3. Use of Agents

Current research into a cental repository of DSSs currently available on the web includes the use of agents as intermediaries between users and individual DSSs. Under this system, known as DecisionNet (Bhargava, King, McQuay, 1995), developers, or DSS providers, would no longer need to be concerned about user interface on the Web, the central site would take care of this, acting as an agent. Developers are required to provide detailed information about the DSS, especially required input and output data

information, so that DecisionNet can build a user interface that can pass data to, and display the results received from, the DSS. Instead of putting the CGI script that executes the DSS on DecisionNet's server, the developer only provides information about the script and what it requires to execute. The area which requires research is developing or implementing a DSS that allows execution by agents.

C.   SUMMARY

Traditionally, DSSs have been limited to the exclusive use of individuals or organizations that had the resources to develop, acquire and maintain them. However, with the increasing popularity of the Web and its widespread use by organizations, the need for multiple copies of a DSS no longer exists. The costs for software acquisition and maintenance are greatly reduced since only one site requires the DSS and maintenance is centralized. As organizations become more familiar with this client-server environment, they should see the benefits of implementing and using a Web based DSS. This research has shown that a DSS developed for stand alone use can be implemented on the Web. There are sufficient tools currently available to provide the basic functionality of a DSS, and as technologies quickly advance, it can only become easier to implement a full featured DSS on the World Wide Web.

APPENDIX A. NEW PROBLEM CGI SCRIPT

One of the features of Delphi is the Object Inspector. This allows the developer to specify properties associated with individual components used in building an application, or this case, a CGI script. Properties defined in the Object Inspector do not appear in the written code but are part of the CGI script, and whatever is defined in the Object Inspector is compiled as part of the script. The first section of each of the Appendicies will indicate the components used and any properties defined in the Object Inspector. The second section will show the actual Pascal code written for the script.

1. COMPONENTS AND THEIR PROPERTIES

Component Used	Property	Defined As
CGIEnvData	no changes required	N/A
Table	DatabaseName	Full path to the table. (i.e. e:\dbd\sarcss)
	TableName	TableName.db (i.e. tinput.db)
DataSource	DataSet	Table1 (refers to Table component)
DdeClientConv	no changes required	N/A
DdeClientItem	DdeConv	DdeClientConv1 (refers to DdeClientConv component)

Table 1. New Problem Script Components and their Properties

There are a total of 67 DdeClientItem components used in the script, but only one DdeClientConv. The properties of each of the DdeClientItem components are all defined the same.

27

2. PASCAL CODE

```pascal
unit Fsarcout;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes,
  Forms, Cgi, DB, DBTables, Cgidb, DdeMan;

type
  Tform1 = class(TForm)
  CGIEnvData1: TCGIEnvData;
  DataSource1: TDataSource;
  Table1: TTable;
  CGIDB1: TCGIDB;
  DDEClientConv1: TDDEClientConv;
  DDEClientItem1: TDDEClientItem;
  DDEClientItem2: TDDEClientItem;
  DDEClientItem3: TDDEClientItem;
  DDEClientItem4: TDDEClientItem;
  DDEClientItem5: TDDEClientItem;
  DDEClientItem6: TDDEClientItem;
  DDEClientItem7: TDDEClientItem;
  DDEClientItem8: TDDEClientItem;
  DDEClientItem9: TDDEClientItem;
  DDEClientItem10: TDDEClientItem;
  DDEClientItem11: TDDEClientItem;
  DDEClientItem12: TDDEClientItem;
  DDEClientItem13: TDDEClientItem;
  DDEClientItem14: TDDEClientItem;
  DDEClientItem15: TDDEClientItem;
  DDEClientItem16: TDDEClientItem;
  DDEClientItem17: TDDEClientItem;
  DDEClientItem18: TDDEClientItem;
  DDEClientItem19: TDDEClientItem;
  DDEClientItem20: TDDEClientItem;
  DDEClientItem21: TDDEClientItem;
  DDEClientItem22: TDDEClientItem;
```

```
DDEClientItem23: TDDEClientItem;
DDEClientItem24: TDDEClientItem;
DDEClientItem25: TDDEClientItem;
DDEClientItem26: TDDEClientItem;
DDEClientItem27: TDDEClientItem;
DDEClientItem28: TDDEClientItem;
DDEClientItem29: TDDEClientItem;
DDEClientItem30: TDDEClientItem;
DDEClientItem31: TDDEClientItem;
DDEClientItem32: TDDEClientItem;
DDEClientItem33: TDDEClientItem;
DDEClientItem34: TDDEClientItem;
DDEClientItem35: TDDEClientItem;
DDEClientItem36: TDDEClientItem;
DDEClientItem37: TDDEClientItem;
DDEClientItem38: TDDEClientItem;
DDEClientItem39: TDDEClientItem;
DDEClientItem40: TDDEClientItem;
DDEClientItem41: TDDEClientItem;
DDEClientItem42: TDDEClientItem;
DDEClientItem43: TDDEClientItem;
DDEClientItem44: TDDEClientItem;
DDEClientItem45: TDDEClientItem;
DDEClientItem46: TDDEClientItem;
DDEClientItem47: TDDEClientItem;
DDEClientItem48: TDDEClientItem;
DDEClientItem49: TDDEClientItem;
DDEClientItem50: TDDEClientItem;
DDEClientItem51: TDDEClientItem;
DDEClientItem52: TDDEClientItem;
DDEClientItem53: TDDEClientItem;
DDEClientItem54: TDDEClientItem;
DDEClientItem55: TDDEClientItem;
DDEClientItem56: TDDEClientItem;
DDEClientItem57: TDDEClientItem;
DDEClientItem58: TDDEClientItem;
DDEClientItem59: TDDEClientItem;
DDEClientItem60: TDDEClientItem;
DDEClientItem61: TDDEClientItem;
DDEClientItem62: TDDEClientItem;
DDEClientItem63: TDDEClientItem;
```

```
      DDEClientItem64: TDDEClientItem;
      DDEClientItem65: TDDEClientItem;
      DDEClientItem66: TDDEClientItem;
      DDEClientItem67: TDDEClientItem;

      procedure FormCreate(Sender: TObject);
   private
      { Private declarations }
   public
      { Public declarations }
   end;

var
   Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.formCreate( sender : TObject );
var
   dUserid : string;
   dAvailAC : string;
   dRadarOvrlap : string;
   dFuelCost : string;
   dExTime : string;
   dODTime : string;
   dRngToSta : string;
   dDegOfCov : string;
   dThrtAxis : string;
   F: Textfile;


begin
   with CGIEnvData1 do
   begin

      { when this program runs under WebSite, parameter #1 will be the
        name of the ini file. }
      webSiteINIFilename := paramstr(1);
      application.onException := cgiErrorHandler;
```

```
application.processMessages;   { need to have this for CGI }

createStdout;
sendPrologue;

{retrieve data from HTML form}
dUserid := getSmallField( 'Userid' );
dAvailAC := getSmallField( 'AvailAC' );
dFuelCost := getSmallField( 'FuelCost' );
dExTime := getSmallField( 'ExTime' );
dODTime := getSmallField( 'ODTime' );
dRadarOvrlap := getSmallField( 'RadarOvrLap' );
dRngToSta := getSmallField( 'RngToSta' );
dThrtAxis := getSmallField( 'ThrtAxis' );
dDegOfCov := getSmallField( 'DegOfCov' );

{send data to temporary table}
Table1.open;
Table1.Delete;
Table1.AppendRecord([dUserid, DateToStr(Date), TimeToStr(Time), dAvailAC,
       dFuelCost, dExTime, dODTime, dRadarOvrlap, dRngToSta, dThrtAxis,
              dDegOfCov]);
Table1.close;

{table was manually linked to the SARCSS spreadsheet when the table created}

{create interim output HTML page on the fly}
{page is created after script has written to output file}
send( '<HTML><HEAD>' );
sendTitle( 'New Problem Confirmation' );
send( '</HEAD><BODY BGCOLOR="#A4C8F0">' );
send( '<Center><H1>SARCSS New Problem<BR>');
send('Input Confirmation</H1></Center>');
send('<HR WIDTH=85% ALIGN=center SIZE=2>');
send('<P>');
send('Thank you for your input!' );
send('<P>');
send('Please go to the');
send('<a href="http://131.120.39.65/sarcss/output.htm"> output page</a>');
send('to view the results.');
send('<HR WIDTH=85% ALIGN=center SIZE=2>');
```

```
      send('<P>');
      sendAddress;
      send( '<P>' );
      send( '</BODY></HTML>' );
      closeStdout;
end;

{create DDE link between spreadsheet and CGI script}
{script is the client to the spreadsheet server}
DDEClientConv1.SetLink('qpw', 'd:\qpw\sarcss\workdss1.wb1');
{assign items to specific cells}
DDEClientItem1.DDEItem := '$K:$A$1..$A$1';
DDEClientItem2.DDEItem := '$K:$A$2..$A$2';
DDEClientItem3.DDEItem := '$K:$A$3..$A$3';
DDEClientItem4.DDEItem := '$K:$A$4..$A$4';
DDEClientItem5.DDEItem := '$K:$A$5..$A$5';
DDEClientItem6.DDEItem := '$K:$B$5..$B$5';
DDEClientItem7.DDEItem := '$K:$C$5..$C$5';
DDEClientItem8.DDEItem := '$K:$A$6..$A$6';
DDEClientItem10.DDEItem := '$K:$B$6..$B$6';
DDEClientItem11.DDEItem := '$K:$C$6..$C$6';
DDEClientItem12.DDEItem := '$K:$A$7..$A$7';
DDEClientItem13.DDEItem := '$K:$B$7..$B$7';
DDEClientItem14.DDEItem := '$K:$C$7..$C$7';
DDEClientItem15.DDEItem := '$K:$A$8..$A$8';
DDEClientItem16.DDEItem := '$K:$A$9..$A$9';
DDEClientItem17.DDEItem := '$K:$B$9..$B$9';
DDEClientItem18.DDEItem := '$K:$C$9..$C$9';
DDEClientItem19.DDEItem := '$K:$A$10..$A$10';
DDEClientItem20.DDEItem := '$K:$B$10..$B$10';
DDEClientItem21.DDEItem := '$K:$C$10..$C$10';
DDEClientItem22.DDEItem := '$K:$A$11..$A$11';
DDEClientItem23.DDEItem := '$K:$A$12..$A$12';
DDEClientItem24.DDEItem := '$K:$B$12..$B$12';
DDEClientItem25.DDEItem := '$K:$C$12..$C$12';
DDEClientItem26.DDEItem := '$K:$A$13..$A$13';
DDEClientItem27.DDEItem := '$K:$B$13..$B$13';
DDEClientItem28.DDEItem := '$K:$C$13..$C$13';
DDEClientItem29.DDEItem := '$K:$A$14..$A$14';
DDEClientItem30.DDEItem := '$K:$A$15..$A$15';
DDEClientItem31.DDEItem := '$K:$B$15..$B$15';
```

```
DDEClientItem32.DDEItem := '$K:$C$15..$C$15';
DDEClientItem33.DDEItem := '$K:$A$16..$A$16';
DDEClientItem34.DDEItem := '$K:$B$16..$B$16';
DDEClientItem35.DDEItem := '$K:$C$16..$C$16';
DDEClientItem36.DDEItem := '$K:$A$17..$A$17';
DDEClientItem37.DDEItem := '$K:$A$18..$A$18';
DDEClientItem38.DDEItem := '$K:$B$18..$B$18';
DDEClientItem39.DDEItem := '$K:$C$18..$C$18';
DDEClientItem40.DDEItem := '$K:$A$19..$A$19';
DDEClientItem41.DDEItem := '$K:$B$19..$B$19';
DDEClientItem42.DDEItem := '$K:$C$19..$C$19';
DDEClientItem43.DDEItem := '$K:$A$20..$A$20';
DDEClientItem44.DDEItem := '$K:$A$21..$A$21';
DDEClientItem45.DDEItem := '$K:$B$21..$B$21';
DDEClientItem46.DDEItem := '$K:$C$21..$C$21';
DDEClientItem47.DDEItem := '$K:$A$22..$A$22';
DDEClientItem48.DDEItem := '$K:$B$22..$B$22';
DDEClientItem49.DDEItem := '$K:$C$22..$C$22';
DDEClientItem50.DDEItem := '$K:$A$23..$A$23';
DDEClientItem51.DDEItem := '$K:$A$24..$A$24';
DDEClientItem52.DDEItem := '$K:$B$24..$B$24';
DDEClientItem53.DDEItem := '$K:$C$24..$C$24';
DDEClientItem54.DDEItem := '$K:$A$25..$A$25';
DDEClientItem55.DDEItem := '$K:$B$25..$B$25';
DDEClientItem56.DDEItem := '$K:$C$25..$C$25';
DDEClientItem57.DDEItem := '$K:$A$26..$A$26';
DDEClientItem58.DDEItem := '$K:$A$27..$A$27';
DDEClientItem59.DDEItem := '$K:$B$27..$B$27';
DDEClientItem60.DDEItem := '$K:$C$27..$C$27';
DDEClientItem61.DDEItem := '$K:$A$28..$A$28';
DDEClientItem62.DDEItem := '$K:$B$28..$B$28';
DDEClientItem63.DDEItem := '$K:$C$28..$C$28';
DDEClientItem64.DDEItem := '$K:$A$29..$A$29';
DDEClientItem65.DDEItem := '$K:$A$30..$A$30';
DDEClientItem66.DDEItem := '$K:$A$31..$A$31';
DDEClientItem67.DDEItem := '$K:$A$32..$A$32';

{link to previously created ASCII file}
AssignFile(F, 'e:\httpd\htdocs\sarcss\output.htm');
{remove previously written data}
Rewrite(F);
```

```
{write results data into the file}
Writeln(F, DDEClientItem1.Text);
Writeln(F, DDEClientItem2.Text);
Writeln(F, DDEClientItem3.Text);
Writeln(F, DDEClientItem4.Text);
Writeln(F, DDEClientItem5.Text);
Writeln(F, DDEClientItem6.Text);
Writeln(F, DDEClientItem7.Text);
Writeln(F, DDEClientItem8.Text);
Writeln(F, DDEClientItem9.Text);
Writeln(F, DDEClientItem10.Text);
Writeln(F, DDEClientItem11.Text);
Writeln(F, DDEClientItem12.Text);
Writeln(F, DDEClientItem13.Text);
Writeln(F, DDEClientItem14.Text);
Writeln(F, DDEClientItem15.Text);
Writeln(F, DDEClientItem16.Text);
Writeln(F, DDEClientItem17.Text);
Writeln(E, DDEClientItem18.Text);
Writeln(F, DDEClientItem19.Text);
Writeln(F, DDEClientItem20.Text);
Writeln(F, DDEClientItem21.Text);
Writeln(F, DDEClientItem22.Text);
Writeln(F, DDEClientItem23.Text);
Writeln(F, DDEClientItem24.Text);
Writeln(F, DDEClientItem25.Text);
Writeln(F, DDEClientItem26.Text);
Writeln(F, DDEClientItem27.Text);
Writeln(F, DDEClientItem28.Text);
Writeln(F, DDEClientItem29.Text);
Writeln(F, DDEClientItem30.Text);
Writeln(F, DDEClientItem31.Text);
Writeln(F, DDEClientItem32.Text);
Writeln(F, DDEClientItem33.Text);
Writeln(F, DDEClientItem34.Text);
Writeln(F, DDEClientItem35.Text);
Writeln(F, DDEClientItem36.Text);
Writeln(F, DDEClientItem37.Text);
Writeln(F, DDEClientItem38.Text);
Writeln(F, DDEClientItem39.Text);
Writeln(F, DDEClientItem40.Text);
```

```
Writeln(F, DDEClientItem41.Text);
Writeln(F, DDEClientItem42.Text);
Writeln(F, DDEClientItem43.Text);
Writeln(F, DDEClientItem44.Text);
Writeln(F, DDEClientItem45.Text);
Writeln(F, DDEClientItem46.Text);
Writeln(F, DDEClientItem47.Text);
Writeln(F, DDEClientItem48.Text);
Writeln(F, DDEClientItem49.Text);
Writeln(F, DDEClientItem50.Text);
Writeln(F, DDEClientItem51.Text);
Writeln(F, DDEClientItem52.Text);
Writeln(F, DDEClientItem53.Text);
Writeln(F, DDEClientItem54.Text);
Writeln(F, DDEClientItem55.Text);
Writeln(F, DDEClientItem56.Text);
Writeln(F, DDEClientItem57.Text);
Writeln(F, DDEClientItem58.Text);
Writeln(F, DDEClientItem59.Text);
Writeln(F, DDEClientItem60.Text);
Writeln(F, DDEClientItem61.Text);
Writeln(F, DDEClientItem62.Text);
Writeln(F, DDEClientItem63.Text);
Writeln(F, DDEClientItem64.Text);
Writeln(F, DDEClientItem65.Text);
Writeln(F, DDEClientItem66.Text);
Writeln(F, DDEClientItem67.Text);
CloseFile(F);
closeApp( application );

end;

end.
```

APPENDIX B. SAVE PROBLEM CGI SCRIPT

1.   COMPONENTS AND THEIR PROPERTIES

Component Used	Property	Defined As
CGIEnvData	no changes required	N/A
Table1	DatabaseName	Full path to the first table. (i.e. e:\dbd\sarcss)
	TableName	TableName.db (i.e. Sarcssin.db)
DataSource1	DataSet	Table1 (refers to Table component)
Table2	DatabaseName	Full path to the second table. (i.e. e:\dbd\sarcss)
	TableName	TableName.db (i.e. sarcsout.db)
DataSource2	DataSet	Table2 (refers to Table component)

Table 2. Save Script Components and their Properties

2.   PASCAL CODE

```
unit Fsarcsav;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes,
  Forms, Cgi, DB, DBTables;

type
  TForm1 = class(TForm)
    CGIEnvData1: TCGIEnvData;
    DataSource1: TDataSource;
    Table1: TTable;
    DataSource2: TDataSource;
    Table2: TTable;
```

37

```pascal
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.formCreate( sender : TObject );
var
  dUserid : string;
  dDate : string;
  dTime : string;
  dAvailAC : string;
  dRadarOvrlap : string;
  dFuelCost : string;
  dExTime : string;
  dODTime : string;
  dRngToSta : string;
  dDegOfCov : string;
  dThrtAxis : string;
  dACReqd : string;
  dTOnSta : string;
  dFEBARng : string;
  dFltTime : string;
  dRngOfCov : string;
  dTotFuelCost : string;

begin

  with CGIEnvData1 do
  begin

      { when this program runs under WebSite, parameter #1 will be the
        name of the ini file. }
```

```
webSiteINIFilename := paramstr(1);
application.onException := cgiErrorHandler;
application.processMessages;  { need to have this for CGI }

createStdout;
sendPrologue;

{Retrieve data from the form}

dUserid := getSmallField( 'Userid' );
dDate := getSmallField('Date');
dTime := getSmallField('Time');
dAvailAC := getSmallField( 'AvailAC' );
dFuelCost := getSmallField( 'FuelCost' );
dExTime := getSmallField( 'ExTime' );
dODTime := getSmallField( 'ODTime' );
dRadarOvrlap := getSmallField( 'RadarOvrLap' );
dRngToSta := getSmallField( 'RngToSta' );
dThrtAxis := getSmallField( 'ThrtAxis' );
dDegOfCov := getSmallField( 'DegOfCov' );
dACReqd := getSmallField( 'ACReqd' );
dTOnSta := getSmallField( 'TOnSta' );
dFEBARng := getSmallField( 'FEBARng' );
dFltTime := getSmallField( 'FltTime' );
dRngOfCov := getSmallField( 'RngOfCov' );
dTotFuelCost := getSmallField( 'TotFuelCost' );

{Add input data to input table}
Table1.open;
Table1.AppendRecord([dUserid, dDate, dTime, dAvailAC, dFuelCost, dExTime,
  dODTime, dRadarOvrlap, dRngToSta, dThrtAxis, dDegOfCov]);
Table1.close;

{Add results to ouput table}
Table2.open;
Table2.AppendRecord([dUserid, dDate, dTime, dACReqd, dFEBARng, dRngOfCov,
  dTOnSta, dFltTime, dTotFuelCost]);
Table2.close;

{Create confirmation HTML page on the fly}
```

```
        send( '<HTML><HEAD>' );
        sendTitle( 'Problem Save Confirmation' );
        send( '</HEAD><BODY BGCOLOR="#A4C8F0">' );
        send('<Center><H1>SARCSS New Problem<BR>');
        send('Save Confirmation</H1></Center>');
        send('<HR WIDTH=85% ALIGN=center SIZE=2>');
        send('<P>');
        send( '<center><Font Size=+1>Your problem has been saved!</Font></center>' );
        send('<p><p><p><center>');
        send('<table border=0 width=300>');
        send('<tr><td><a href="http://131.120.39.65/sarcss/newpg3-1.htm">New
                Problem</a></td>');
        send('<td><a href="http://131.120.39.65/sarcss/sarquery.htm">Retrieve
                Problem</a></td>');
        send('<td>
                <a href="http://bhargava.as.nps.navy.mil/dNethome.html">Exit</a></td></tr>
                </table></center>');

        send('<HR WIDTH=85% ALIGN=center SIZE=2>');
        send('<p>');
        sendAddress;
        send( '</BODY></HTML>' );
        closeStdout;
    end;

    closeApp( application );

end;

end.
```

APPENDIX C. PROBLEM RETIEVAL CGI SCRIPT

1. COMPONENTS AND THEIR PROPERTIES

Component Used	Property	Defined As
CGIEnvData	no changes required	N/A
Table	DatabaseName	Full path to the table. (i.e. e:\dbd\sarcss)
	TableName	TableName.db (i.e. tretriev.db)
DataSource1	DataSet	Table1 (refers to Table component)
Query1	no changes required	N/A
DataSource2	DataSet	Query1 (refers to Query component)
Query2	no changes required	N/A
DataSource3	DataSet	Query2 (refers to Query2 component)
CGIDB1	CGI	CGIEnvData1
	DataSource	DataSource1
CGIDB2	CGI	CGIEnvData1
	Caption	Input Parameters
	DataSource	DataSource2
	FieldDisplayList	[TStringList]
	useFieldDisplayList	True
CGIDB3	CGI	CGIEnvData1
	Caption	Problem Results
	DataSource	DataSource3
	FieldDisplay List	[TStringList]
	useFieldDisplayList	True

Table 3. Problem Retieval Script Components and their Properties

The TStringLists for CGIDB2 and CGIDB3 are defined at script creation. The lists are composed of specific fields from the input and output tables that will be displayed to the user when the script creates the output HTML page on the fly. These lists are used to define which fields in the tables will be displayed when the script creates the output HTML page on the fly. The list for the input table includes the following fields: Date, Time, Available Aircraft, Fuel Cost, Exercise Time, On Deck Time, Radar Overlap, Range to Station, Threat Axis, Degree of Coverage. The list for the results table includes the following fields: Date, Time, Aircraft Required, FEBA Range, Range of Coverage, Time on Station, Flight Time Per Aircraft, Total Fuel Cost.

2. PASCAL CODE

```
unit Fsarq4;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes,
  Forms, Cgi, DB, DBTables, Cgidb;

type
  TForm1 = class(TForm)
    CGIEnvData1: TCGIEnvData;
    CGIDB1: TCGIDB;
    DataSource1: TDataSource;
    Query1: TQuery;
    CGIDB2: TCGIDB;
    DataSource2: TDataSource;
    Query2: TQuery;
    CGIDB3: TCGIDB;
    DataSource3: TDataSource;
    Table1: TTable;
    procedure FormCreate(Sender: TObject);
```

```
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.formCreate( sender : TObject );
var
  sUserid : string;
  sDate : string;
  sTime : string;
  sAvailAC : string;
  sFuelCost : string;
  sExTime : string;
  sODTime : string;
  sRadarOvrlap : string;
  sThrtAxis : string;
  sDegOfCov : string;
  sACReqd : string;
  sFEBARng : string;
  sRngOfCov : string;
  sTimeOnSta : string;
  sFltTime : string;
  sTotFuelCost : string;

begin

  with CGIEnvData1 do
  begin

    { when this program runs under WebSite, parameter #1 will be the
      name of the ini file. }
    webSiteINIFilename := paramstr(1);
    application.onException := cgiErrorHandler;
```

```
application.processMessages;   { need to have this for CGI }

createStdout;
sendPrologue;

{Get userid from the HTML form}
sUserid := getSmallField( 'Userid' );

{Save userid in temporary table}
Table1.open;
Table1.delete;
Table1.appendRecord([sUserid]);

{Query input table using userid}
Query1.close;
Query1.sql.clear;
Query1.sql.add('Select * from "e:\dbd\sarcss\sarcssin.db" where Userid = "'+ sUserid
     +'"');
Query1.open;

{Query output table using userid}
Query2.close;
Query2.sql.clear;
Query2.sql.add('Select * from "e:\dbd\sarcss\sarcsout.db" where Userid = "'+ sUserid
     +'"');
Query2.open;

{Create output HTML page on the fly}

send( '<HTML><HEAD>' );
sendTitle( 'SARCSS Problem Retrieval' );
send( '</HEAD><BODY BGCOLOR="A4C8F0">' );
send('<Center><H1>SARCSS Problem Retrieval</H1></Center>');
send('<Center>');
send('<HR WIDTH=85% ALIGN=center SIZE=2><P>');
send( '<Font Size=+2>Your previous problem(s)</Font>' );
send('</center>');
send('<P><Center>');
CGIDB1.drawTable;   {creates userid table on HTML page}
send('<P>');
CGIDB2.drawTable;   {creates input table on HTML page}
```

```
        send('<P>');
        CGIDB3.drawTable;   {creates output table on HTML page}
        send('</Center><P>');

        send('<Center><Table border=0>');
         send('<tr><td><a href="http://131.120.39.65/sarcss/sarmenu.htm">Main
Menu</a></td>');
        send('<td></td><td><a href="http://131.120.39.65/sarcss/sarprob1.htm">New
         Problem</a></td>');
        send('<td></td><td>
                <a href="http://bhargava.as.nps.navy.mil/dNethome.html">Exit</a></td></tr>');
        send('</table></center><p>');
        sendHR;
        sendAddress;
        send( '</BODY></HTML>' );
        closeStdout;
    end;

    closeApp( application );

end;

end.
```

LIST OF REFERENCES

Braley, Christopher L., *Optimizing Aircraft Radar Coverage: A Decision Support System Prototype*, Naval Postgraduate School MS Thesis, September 1995.

Bhargava, H. K., A. S. King, and D. S. McQuay, "DecisionNet: Modeling and Decision Support Over the World Wide Web", *Proceedings of the Third ISDSS Conference, Hong Kong*, June 1995.

Bhargava, H. K., R. Krishnan, R. Muller, *Decision Support on Demand: Emerging Electronic Markets for Decision Technologies*, February 1996.

Denny, Robert B., "Windows CGI 1.3a Interface", *http://solo.dc3.com/wsdocs/32demo/windows-cgi.html*.

Gonzalez, Sean, "Scripting the Web", *PC Magazine (Network Edition)*, February 6, 1996, Vol._15, No. 3, 263-265.

Goodyear Tire and Rubber Company, *http://www.goodyear.com*.

Long, Larry, *Introduction to Computers and Information Systems*, Fourth Edition, Englewood Cliffs, NJ: Prentice-Hall, Inc. 1994, 208.

Lynworth, Ann, "Intro to CGI and Delphi", *http://www.href.com/cgi-win/hubws.exe?HREFINFO:filelern:236255:*.

National Center for Supercomputing Applications, "CGI: Common Gateway Interface", *http://hoohoo. ncsa.uiuc.edu/cgi/intro.html/*.

Sprague, Ralph H. Jr., and Barbara C. McNurlin, *Information Systems Management in Practice*, Third Edition, Englewood Cliffs, NJ: Prentice-Hall, Inc. 1993, 381.

World Wide Web Consortium, "Overview of CGI", *http://www.w3.org/pub/WWW/CGI/Overview.html*.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2.	Dudlley Knox Library, Naval Postgraduate School 411 Dyer Rd. Monterey, CA 93943-5101	2
3.	Professor Hemant K. Bhargava (Code SM/BH) Naval Postgraduate School Monterey, CA 93943-5002	2
4.	Professor Suresh Sridhar (Code SM/SR) Naval Postgraduate School Monterey, CA 93943-5002	2
5.	LCDR Patrick E. Protacio Commander Naval Air Force U. S. Atlantic Fleet 1279 Franklin St. Norfolk, VA 23511-2494 Code: N6	1
6.	Arthur S. Protacio 13654 Kingsman Rd. Woodbridge, VA 22193	1
7.	James Quinones 29 Mandrake Way Irvine, CA 92715-2713	1